3. Retrieve the Source Code

Created: April 1, 2003
Updated: February 20, 2004

Overview

The overview for this chapter consists of the following topics:

- Introduction
- Chapter Outline

Introduction

The first step in working with the C++ Toolkit is obtaining the source code. Source code can be obtained using CVS for in-house users or FTP for external users. This chapter describes how to setup the CVS client to obtain the source code. In addition, this chapter describes the use of scripts that can simplify obtaining only the necessary source code components.

Chapter Outline

The following is an outline of the topics presented in this chapter:

CVS Source Code Retrieval for Public Read-only Access

The public CVS server is available, which contains the public part of the C++ Toolkit. To use it, follow exactly the in-house CVS usage instructions with two exceptions:

1. The `CVSROOT` env. variable should be set to:
```
:pserver:anoncvs@anoncvs.ncbi.nlm.nih.gov:/vault
```
2. Use empty password to login:
```
> cvs login
     Logging in to :pserver:anoncvs@anoncvs.ncbi.nlm.nih.gov:/vault
     CVS password:  <just press ENTER here>
```

For details on checking out the production CVS tree see the section Checking Out the Production NCBI C++ Toolkit Source Tree.

CVS Source Code Retrieval for In-House Users with Read-Write Access

For a detailed description of the CVS utility see the CVS online manual or run the commands `"man cvs"` or `"cvs --help"` on your Unix workstation.

The following is an outline of the topics presented in this section. Select the instructions appropriate for your **development** environment.

- NCBI CVS Tree Contents
- Setting up CVS client
- CVS Retrieval under Unix
    - CVS Retrieval of the C++ Toolkit
        - Checking Out the Internal NCBI C++ Toolkit Source Tree
        - Checking Out the Production NCBI C++ Toolkit Source Tree
        - cvs_core.sh: Retrieving core components
            - cvs_core.sh: Retrieve Only the Portable and Core Components
                - Path Note for Scripts
            - cvs_core.sh Arguments
            - Contents of the Portable Core Source Tree
            - Supplement Contents Using the cvs_core.sh Options
            - cvs_core.sh MSVC++-related Options

NCBI CVS Tree Contents

The NCBI C++ Toolkit CVS repository contains all source code, scripts, utilities, tools, tests and documentation required for a variety of builds (e.g., "Debug" and "Release" versions of the Toolkit) under the major classes of operating system (`Unix`, `MS Windows` and `Mac OS`). The CVS tree has the same global and project-level organization of the source tree yet contains the code needed for all supported builds.

Setting up CVS client

CVS client installation instructions are available on separate pages for `MS Windows` and `Mac OS` systems. Here are the instructions for setting up CVS client on UNIX:

1. Set `CVSROOT` env.variable to: *:pserver:${LOGNAME}@cvsvault:/src/NCBI/vault.ncbi*. Note that for NCBI Unix users, this may already be set if you specified *developer* for the *facilities* option in the *.ncbi_hints* file in your home directory.
2. Run the command: `cvs login` You will be asked for a password -- enter the word **allowed**. This command will record your login info into *~/.cvspass* file (so you would not have to login into CVS in the future, ever). **You may need to create an empty***~/.*

*cvspass***file before logging in as some CVS clients apparently just cannot create it for you.If you get an authorization error, then send e-mail with the errors to** cpp-core.

3. If you have some other CVS snapshot which was checked out with old value of CVS-ROOT, you should commit all your changes first, then delete completely the old snapshot dir and run: `cvs checkout` to get it with new `CVSROOT` value.

4. Now you are all set and can use all usual CVS commands.

NOTE: When you are in a directory that was created with `cvs checkout` by another person, a local *./CVS/* subdirectory is also created in that directory. In this case, the cvs command ignores the current value of the `CVSROOT` environment variable and picks up a value from *./CVS/Root* file. Here is an example of what this *Root* file looks like:

```
:pserver:username@cvsvault:/src/NCBI/vault.ncbi
```

Here the *username* is the user name of the person who did the initial CVS checkout in that directory. So CVS picks up the credentials of the user who did the initial check-in and ignores the setting of the `CVSROOT` environment variable, and therefore the CVS commands that require authorization will fail. There are two possible solutions to this problem:

1. Create your own snapshot of this area use the `cvs get` command.
2. Impersonate the user who created the CVS directory by creating in the *~/.cvspass* file another string which is a duplicate of the existing one, and in this new string change the username to that of the user who created the directory. This hack will allow you to work with the CVS snapshot of the user who created the directory. However, this type of hack is not recommended for any long term use as you are impersonating another user.

CVS Retrieval under Unix

The following topics are discussed in this section:

- CVS Retrieval of the C++ Toolkit
    - Checking Out the Internal NCBI C++ Toolkit Source Tree
    - Checking Out the Production NCBI C++ Toolkit Source Tree
    - cvs_core.sh: Retrieving core components
        - cvs_core.sh: Retrieve Only the Portable and Core Components
            - Path Note for Scripts
        - cvs_core.sh Arguments
        - Contents of the Portable Core Source Tree

CVS Retrieval of the C++ Toolkit

This section discusses the methods of checking out the entire CVS tree or just the necessary portions. An important point to note is that the whole NCBI C++ tree is very big because it contains a lot of internal user projects. There are also numerous platform-specific files, and even whole sub-trees, which you will never need unless you work on other platforms. Therefore it is frequently sufficient, and in fact, usually advisable, to retrieve only those files of direct interest using the shell scripts in the *$NCBI/c++/scripts* directory.

The auxiliary scripts ***cvs_core.sh*** and ***cvs_core.bat*** checkout only the core (basic and portable) NCBI C++ Toolkit sources for a desired platform. A similar auxiliary script exists to import the source from a single project (***import_project.sh***). To facilitate the creation of a new project, use the script (***new_project.sh***), which generates new directories and makefiles for the new project from templates. The script also checks out a specified sample application from the CVS tree that may be adapted for the new project or built directly as a demonstration.

The following additional topics are discussed in this section:

        - cvs_core.sh Arguments
        - Contents of the Portable Core Source Tree
        - Supplement Contents Using the cvs_core.sh Options
        - cvs_core.sh MSVC++-related Options
        - MSVC++ Project File Conversion Tools
            - Shell Scripts
            - Transition to single-configuration project files
    - import_project.sh: Retrieve Source for an Existing Project
    - update_core.sh: Update the Portable and Core Components
    - update_projects.sh: Update Source for Specific Projects

Checking Out the Internal NCBI C++ Toolkit Source Tree

You can checkout the entire internal NCBI C++ source tree from CVS to your local directory (e.g., *foo/c++/*) just by running:

```
cd foo
cvs checkout -d c++ internal/c++
```

Alternatively, you can get a copy of the sources using the CVS `get` command:

```
cvs get -d c++ internal/c++
```

Caution: Be aware that sources checked out through the internal CVS tree have the latest sources and are different from the public release that is done at periodic intervals. As such these sources are relatively unstable, "development" sources, and are not guaranteed to work properly or even compile. Use these sources at your own risk (and/or to apply patches to stable releases). The sources are usually better by the end of day and especially by the end of the week (like Sunday evening).

Checking Out the Production NCBI C++ Toolkit Source Tree

Besides the internal NCBI C++ source tree, there is the C++ Toolkit "production" source tree that has been added to the public CVS server area. This tree contains stable snapshots of the "development" C++ Toolkit tree. Please note that these sources are lagging behind, sometimes months behind the current snap-shot of the sources.

You can checkout the whole "production" NCBI C++ source tree from the public CVS server area to your local directory just by running:

```
cvs checkout  -r ProductionBuild_YYYYMMDD  -d c++  production/c++
```

or

```
cvs get   -r ProductionBuild_YYYYMMDD  -d c++  production/c++
```

where symbolic tag *ProductionBuild_YYYYMMDD* is listed in the corresponding "snapshot-info" file located under *c++/patches/*

For example, there is only one "snapshot-info" file, "ORIGINAL.txt", and it lists the tag as:

```
cvs checkout  -r ProductionBuild_20031212  -d c++  production/c++
```

or

```
cvs get  -r ProductionBuild_20031212  -d c++  production/c++
```

Caution: If you checkout the current state of the "production" C++ Toolkit tree (i.e. if you do not use the sym.tag), then there is a risk that you get unbuildable set of sources.

cvs_core.sh: Retrieving core components

The following topics are discussed in this section:

- cvs_core.sh: Retrieve Only the Portable and Core Components
    - Path Note for Scripts
- cvs_core.sh Arguments
- Contents of the Portable Core Source Tree
- Supplement Contents Using the cvs_core.sh Options
- cvs_core.sh MSVC++-related Options
- MSVC++ Project File Conversion Tools
    - Shell Scripts
    - Transition to single-configuration project files

cvs_core.sh: Retrieve Only the Portable and Core Components

Usage Summary (path note):

```
cvs_core.sh <dir> [--with-<feature>]... [--without-<feature>]... [--date=<date>]  [--
<platform>] [--<cvstree>]
```

Default settings:

```
cvs_core.sh <dir> --with-cvs --date=<current> -all  --internal
```

The arguments to ***cvs_core.sh*** are discussed in the ***cvs_core.sh*** Arguments subsection.

For the major three platforms, a typical invokation for "core" source retrieval is shown below:

```
Unix:

cvs_core.sh <dir> --unix

MS-Win:

cvs_core.sh <dir> --msvc --with-objects --without-dizzy

MacOS:

cvs_core.sh <dir> --mac --with-objects
```

Note:*dizzy* is the name of a Microsoft Terminal Server on the internal NCBI network and as such the *--with-dizzy* option would apply to internal NCBI users only.

The NCBI C++ Toolkit CVS repository contains the source for many features and extension beyond the core of portable functionality. Often one wants to obtain a set of Toolkit sources that are free of the non-portable elements, and the ***cvs_core.sh*** script performs this task across the range of supported platforms. Options to the basic command allow the developer to further tailor the retrieved source by including (or excluding) portions of the Toolkit not checked out by default.

Path Note for Scripts

If the Toolkit is installed on your machine at $NCBI, these scripts should be invoked as:***$NCBI/c++/scripts/<script_name>*** if that directory is not in your $PATH.

cvs_core.sh Arguments

Table 1 describes the arguments of ***cvs_core.sh***, along with their default values. Only the target directory is mandatory. The optional *--with/without-<feature>* argument pairs include or exclude portions of the Toolkit from the checked-out source. While both members of the pair may appear on the command line, only the final one controls the script's behavior. Certain settings are meaningful only for certain <*platform*>s (most often with respect to MS Windows platforms).

Table 1. *cvs_core.sh* Arguments

Argument	Description	Permitted Values
<dir>	Path to where tree will be checked out. This argument is **required**.	Valid writable directory name (cannot exist already); name cannot start with "-"
<platform>	Obtain sources for the specified platform(s).	*unix* (Unix systems) *msvc* (for Microsoft Visual C++ environment) *mac* (Macintosh systems) *cygwin* (Red Hat's Cygwin UNIX environment for MS Windows) *all* (all systems) ***default: all***
--with-cvs	Include CVS-related directories in the resultant tree.	Even if this flag is not present, it defaults to **ON** which means that the CVS-related directories are in the resultant tree (unless overriden by *--without-cvs* flag).
--without-cvs	Exclude CVS-related directories from the resultant tree.	If not present, this flag has no affect on the behavior of *--with_cvs*. If present, it excludes CVS-related directories from the resultant tree, and instead does a CVS "export" as of the date specified by the date parameter.
--date	Checkout as of the specified date.	If the *--date* flag is missing the current timestamp is used.
--with-objects	Generate ASN.1 serialization code in the *objects/* directory.	If the *--with-objects* flag is present, the objects, object manager and object tools are checked out and serialization code is generated from the

Argument	Description	Permitted Values
		ASN.1 specifications. If the --*with-objects* flag is not present, the objects, object manager and object tools are still checked out (unless overriden by the --*without-objects* flag) but no serialization code is generated. If the --*without-objects* flag is present, then the object componenets will not be checked out. (Unix platforms: the code generation can be done later, during the build)
--*without-objects*	Do not check out the objects, object manager or object tools directory or generate ASN.1 serialization code.	If not present, this flag has no affect and the behavior for --*with-objects* applies. That is, unless explicitly turned off by providing this argument the objects, object manager and object tools are always checked out. The main purpose of this flag is to ensure that the object components are not checked out.
--*with-ctools*	Checkout core projects responsible for working together with the NCBI C Toolkit (*ctools/* directory).	If not present, it defaults to still checking out the *ctools/* directory unless overriden by the --*without-ctools* flag.
--*without-ctools*	Do not checkout core projects responsible for working together with the NCBI C Toolkit.	If not present, this flag has no affect and the behavior for --*with-ctools* applies. That is, unless explicitly turned off by providing --*without-ctools*, the ctools are always checked out. The main purpose of this flag is to ensure that the ctools components are not checked out.
--*with-gui*	Checkout projects that depend on wxWindows.	If not present, it defaults to still checking out the *gui* components unless overriden by the --*without-gui* flag.
--*without-gui*	Do not checkout projects that depend on wxWindows.	If not present, this flag has no affect and the behavior for --*with-gui* applies. That is, unless explicitly turned off by providing --*without-gui*, the gui components are always checked out. The main purpose of this flag is to ensure that the gui components are not checked out.

Argument	Description	Permitted Values
--*<cvstree>*	Whether to include internal NCBI components of the CVS tree.	The --*<cvstree>* is replaced by either **--production** or **--internal**. If **--internal** is specified, the internal CVS tree branch for NCBI users is checked out. If **--production** is specified, the production CVS tree branch for non-NCBI users is checked out. If not specified, this option defaults to **--internal**.
The following have an affect only for <*platform*>={*msvc,all*}		
--*with-vc-cfgs [=<selection>]*	Get multi-configuration Microsoft Visual C++ project files.	This can take three values: *all* (get all project files), *inhouse* (get only the NCBI inhouse project files) and *none* (get no project files). The default is *all*.
--*without-vc-cfgs*	Do not get multi-configuration Microsoft Visual C++ project files.	*Default: OFF* (***ON*** is the same as --*with-vc-cfgs=none*) Note:--*without-vc-cfgs* is not compatible with --*without-dizzy*
--*with-dizzy*	Retain paths to \\DIZZY\ in Microsoft Visual C++ project files.	*default: OFF*. Note that \\DIZZY is the name of a Microsoft Terminal Server accessible to internal NCBI users. If this option is not specified, the default of OFF means that there wont be paths to \\DIZZY within the project file.
--*without-dizzy*	Remove paths to \\DIZZY\ in Microsoft Visual C++ project files.	*default: OFF* Note: --*without-dizzy* is not compatible with --*without-vc-cfgs*. \\DIZZY is the name of a Microsoft Terminal Server accessible to internal NCBI users.

Contents of the Portable Core Source Tree

In this section, all paths and filenames are in the CVS tree and unless otherwise specified, should be taken as relative to the common root *$CVSROOT/internal/c++*.

Common Source: Table 2 lists those which are always recursively checked out, regardless of the arguments to ***cvs_core.sh***:

Table 2. List of directories always checked out

doc	
include/corelib	*src/corelib*
include/connect	*src/connect/test*

include/serial	*src/serial*
include/cgi	*src/cgi*
include/html	*src/html*
include/util	*src/util*
include/hello	*src/hello*
include/app	*src/app*

Supplement Contents Using the *cvs_core.sh* Options

Platform-Specific Source In addition to the above common source, the various <*platform*> options will populate the remainder of the checked-out CVS source tree differently. At present a laundry-list for each platform option is not provided here, however you may view the source to examine the most current status. In general, the platform and argument-sensitive parts of the source reside in the directories shown in Table 3.

Table 3. Directories containing argument sensitive parts

compilers	
connect/daemons	
connect/mitsock	
include/dbapi	*src/dbapi*
scripts	

Feature-Specific Source The --*with*/--*without*-<*feature*> options enable or inhibit portions of the CVS tree from being included in the checked-out out source tree. Consult Table 4 to identify where a certain option affects the source tree.

Table 4. Directories affected by enabling/disabling the --with/--without option

Feature	Affected Directories
cvs	*/CVS*
ctools	*include/ctools, src/ctools, compilers*
dizzy	*compilers*
gui	*include/gui, src/gui, compilers*
objects	*include/objects/*, src/objects/*, compilers, scripts, src*
vc-cfgs	*compilers*

cvs_core.sh MSVC++-related Options

A project file, checked out by script ***cvs_core.sh***, is by default expanded to contain all 6 configurations (*Debug*, *DebugMT*, *DebugDLL*, *Release*, *ReleaseMT*, *ReleaseDLL*). However, for a developer this might not be very convenient, as the resulting project file (if modified) cannot be checked in back to the CVS repository. Therefore, we provide an option (to be used by Toolkit developers), which allows to check out C++ Toolkit "as is", i.e. without expanding the project files:

```
$ cvs_core.sh ... --without-vc-cfgs
```

Note that in this case only one configuration, namely *DebugDLL*, will be available in the projects, and the batch file ***all.bat*** (even if started without arguments) will build only this one configuration.

Suppose that you want to build only one configuration (e.g. *ReleaseDLL*). You can do it in either of the following ways:

1. Checkout the Toolkit sources using ***cvs_core.sh*****without** option --*without-vc-cfgs*.
2. Move entire Toolkit tree to your PC.
3. Run "`all.bat ReleaseDLL`" to build the Toolkit. In this case you are able to build other configuration(s) as well (e.g. "`all.bat Debug`").

or:

1. Checkout the Toolkit sources using ***cvs_core.sh*****with** option --*without-vc-cfgs*.
2. Apply script ***one2one.sh*** recursively for all project files:
   ```
   $ find . -name "*.dsp" -exec ./one2one.sh {} ReleaseDLL \;
   ```
3. Move entire Toolkit tree to your PC.
4. Run "***all.bat***" to build the Toolkit.

Another recently added MSVC-specific option is --*without-dizzy*, which is propagated to the script ***one2all.sh***, called to build 6 configurations in every project file; therefore, this option cannot be combined with --*without-vc-cfgs*.

Note that although is it acceptable to specify default behavior with --*with-vc-cfgs* and --*with-dizzy*, we strongly recommend not to use these forms of the options.

MSVC++ Project File Conversion Tools

This manual assumes that the UNIX-like environment is used to prepare and manage MSVC++ project files, which in turn can be used to build NCBI C++ Toolkit by MSVC++ Developer Studio. The following tools must be available in order for the conversion scripts to work: *cat*, *cp*, *diff*, *echo*, *egrep*, *expr*, *find*, *grep*, *head*, *mv*, *rm*, *sh*, *tail*, *test*, *touch*, and *tr*.

The idea behind the automatic project file conversion is that usually developer works only with some fixed configuration (e.g. debug, single-threaded), yet he/she must keep all other configurations in-sync. Hereby we provide the developer with the means to work with (and make changes to) a single configuration at any given moment of time. Then, when the build process is about to occur, all other deficient configuration(s) can be generated automatically from that fixed (template) configuration.

There are some limitations to the above scheme:

- The per configuration dependencies are not supported, they are ignored and excluded from the resulting project file, with warning printed.

- If libraries or object files are explicitly included in the project file list, they usually refer to fixed file placements (e.g. *src\connect\DebugMT\ncbi_socket.obj*), i.e. bound to the particular configuration it was built with. Therefore, you must **never** use the explicit inclusion of this kind when creating your single-configuration (template) project files. Instead, use workspace file (*.dsw*) *dependency mechanism* to link with such libraries and object files, and use linker options (library list) without explicit paths.

The following additional topics are discussed in this section:

- Shell Scripts
- Transition to single-configuration project files

Shell Scripts

All the work of managing the project files is done by using of one or more of the following shell scripts:

- ***all2one.sh*** - looks through the given project file, prints out available configurations, and allows to extract any of those. If there is only one configuration, the script does nothing. Special care is taken to first check the project file consistency to avoid corrupting the vital project file structure. Note, however, that the warning is printed if the project file contains per configuration issues, which cannot be handled in a graceful way, and thus ignored. Input file can be in either DOS or UNIX text file format (respectively, with or without CR (carriage return) incorporated before LF (life feed) at line ends). Output file is always in UNIX text file format, which is suitable for CVS repository. **Note:** Aside the project file name, the second argument can be given to the script. This argument is used as a name of configuration to extract without interactive inquiry. If the specified configuration is not found, the script aborts with an error message.

- ***one2all.sh*** - takes a single-configuration project file (perhaps the one created by ***all2one.sh*** and derives multi-configuration project file containing 3 production configurations: `Release`, `ReleaseMT`, and `ReleaseDLL`; and 3 debug configurations:`Debug`, `DebugMT`, and `DebugDLL`. One option, *--without-dizzy*, can precede the file name. If specified, then the C compiler, resource compiler and linker options, which refer to additional paths, involving *\\DIZZY*, are removed. Output file is always produced in DOS text file format (that is suitable for immediate consumption by MSVC++), whereas input file can be in either UNIX or DOS format. **Note:** There is one additional argument, which is reserved for internal use only. When given as *2*, only `DebugDLL` and `ReleaseDLL` configurations are placed to the output file. When given as *4*, only the following four configurations appear: `Debug`, `DebugMT`, `Release`, and `ReleaseMT`.

- ***one2one.sh*** - takes two arguments: single-configuration project file name and one of standard configuration names (`Debug`, `DebugMT`, `DebugDLL`, `Release`, `ReleaseMT`, `ReleaseDLL`) and replaces the given project file with a new one, having the requested configuration. In case of non-standard configuration name, an error results. Being an 'interface' script to both ***all2one.sh*** and ***one2all.sh***, this scipt can accept one option, *--without-dizzy*, as described above. Output file is created in UNIX text file format.
- ***include.sh*** - takes an optional argument list, each entry of which is a relative path to include file directory (default is the standard NCBI C++ Toolkit include directory). The script then recursively looks through all project files, and tries to update compiler switches with the given directory list. Directories, already specified in project files, are either replaced (when the new directory resembles the old one), or added to the current list of include directories. The main purpose of the script is to eliminate "global" configuration through "Tool/Options", which is otherwise required to specify include file directories. Generated paths to include files are always relative to the project file. The script is to be used from the *msvc_prj* directory each time, when the project file subtree is a kind rearranged and must be updated in the CVS. Therefore, the project files are always produced in UNIX text file format.
- ***commit.sh*** - control script for use from inside the CVS upon each commit to ensure that the project being checked in contains one fixed configuration only (namely, `DebugDLL`). This script as well checks that the file being committed is in UNIX text file format, and rejects if not. The extensions (and only in directories specified in *DIRLIST* parameter, as described below) checked are:*.dsw*, *.dsp*, *.bat*. No explicit call to this script is ever necessary: it is entirely internal to CVS and transparent for the user. **Additional** functionality has been added to this script: if it detects that the file being committed contains paths either to standard NCBI C Toolkit include files/libraries, or wxWindows include files/libraries, then the paths are expanded to specify both relative (i.e. local) and absolute (i.e. via *\\DIZZY \public\ncbi*) locations, thus making possible to build the project using either local or prebuilt libraries. We require that if a toolkit is used locally (i.e. **not** via *\\DIZZY\public\ncbi*), then the upper-level directory of the toolkit has to be placed at the same level as the NCBI C++ Toolkit top-level directory. We reserve that *wxwin* and *ncbi* subdirectories, when appear in the same parent directory as *cxx*, contain wxWindows and NCBI C Toolkit correspondingly. Click here to know how to use the wxWindows package in these requirements. One can later remove absolute references to *\\DIZZY* using option *--without-dizzy* with either ***one2all.sh***, or ***one2one.sh***.

Please note that the above tools are **not** project file *generation* tools, but rather the project file *conversion* tools. That is, the resulting project file is correct if and only if the source project file is. When a project file gets converted to a single-configuration template, all the compiler switches and definitions, as well as other vital environment, have to be defined in a right way.

DOS batch file ***all.bat*** can generate all projects in a batch mode, provided that *msdev.exe* is in your `PATH`. As an argument to the batch file you can specify either *ALL* (or no arguments at all), or any combination of `Debug`, `DebugMT`, `DebugDLL`, `Release`, `ReleaseMT`, `ReleaseDLL` to build either all available, or particular configuration(s) respectively. You can always use MSVC++ Developer Studio (just load workspace file *ncbi_cpp.dsw*) to build the Toolkit in the interactive manner.

Transition to single-configuration project files

This step applies only once, when all current project files are converted to single-configuration ones, which later can be automatically expanded back to multi-configuration project files, as required for (production and/or regular) builds.

1. Check out the MSVC++ project files and tool scripts from *internal/c++/compilers/msvc_prj* directory:
```
$ mkdir ~/msvc_prj; cd ~/msvc_prj
$ cvs co -d . internal/c++/compilers/msvc_prj
```
2. Apply the following command to convert multi-configuration project files to single-configuration project files:
```
$ find . -name "*.dsp" -print -exec ./all2one.sh {} \;
```
   For each project file choose the configuration (among available), which was tested, and known to work. Note that many project files were created automatically by MSVC++, and contain 2 or more configurations, but in most cases only one (notably: `Debug`) was customized to work and to build this particular project, the others are just dummies.
3. Be sure that the process completed successfully and then delete backup copies:
```
$ find . -name "*.dsp.bak" -exec rm -f {} \;
```
4. You may wish to use the script ***one2one.sh*** for converting all project files to standard single configuration `DebugDLL`, if configurations, different from `DebugMT`, have been chosen in step 2 above:
```
$ find . -name "*.dsp" -print -exec ./one2one.sh {} DebugMT \;
```
5. Check out *CVSROOT* directory (this is an administrative directory used to control the entire CVS tree):
```
$ mkdir ~/myCVSROOT; cd ~/myCVSROOT
$ cvs co CVSROOT
```

6. Chdir to the checked out copy of *CVSROOT*:
```
$ cd CVSROOT
```

7. Install (by copying and giving execution permissions) the following scripts in *CVSROOT* directory:
```
$ ( cd internal/c++/compilers/msvc_prj; cp commit.sh all2one.sh ~/myCVSROOT )
$ chmod a+x commit.sh all2one.sh
$ cat >> checkoutlist
  commit.sh
  all2one.sh
<press Ctrl-D>
$ cat >> commitinfo
$ CVSROOT/CVSROOT/commit.sh
<press Ctrl-D>
```

8. Make sure that the files *checkoutlist* and *commitinfo* contain only one entry per each shell script, delete any other entries if they exist.

9. Edit script ***commit.sh*** so that the line starting from *DIRLIST*= would contain a list of directories, delimited by spaces, relative to CVS tree origin, and which must be checked against for having single-configuration project files only.

10. Now check in the changes made in *CVSROOT* directory back to the CVS repository:
```
$ cvs add commit.sh all2one.sh
$ cvs commit -m "MSVC++ Project File Control Added"
```

    You should see message, saying that the CVS administrative directory is being rebuilt. Erase local copy of *CVSROOT* directory.

11. Now chdir back to checked out copy of directory *internal/c++/compilers/msvc_prj*, which now should contain single-configuration projects only. You can commit the projects back to the CVS. This is a bit tricky, as formerly the project files were kept in binary form, and now they have to be stored in a native text file format. The easiest way of workaround is to do the following:
```
$ ( cd $CVSROOT/internal/c++/compilers; mv msvc_prj msvc_prj.old )
$ find . -name CVS -exec rm -rf {} \;
$ cvs import -m "New projects" internal/c++/compilers/msvc_prj NCBI Exp
```

12. Erase the local copy of project file directory. If you wish to work further with projects, you should re-check them out again to ensure that everything is connected back to the repository in a right way.

import_project.sh: Retrieve Source for an Existing Project

Usage Summary (path note):
```
import_project.sh <cvs_tree_path> [builddir]
```

In many cases, you work on your own project which is a part of the NCBI C++ tree, and you do not want to check out, update and rebuild the whole NCBI C++ tree. -- Instead, you just want to use headers and libraries of the pre-built NCBI C++ Toolkit to build your project.

The shell script ***import_project.sh*** will checkout your project's *src* and *include* directories from CVS, and it will create a (temporary) makefile based on the project's customized makefile. This makefile will also contain a reference to the pre-built NCBI C++ Toolkit.

For EXAMPLE:

```
import_project.sh hello
```

will check out the whole `hello` demo project from the NCBI C++ tree (*internal/c++/{src, include}/hello/*), and create a makefile *Makefile.hello_app* that uses the project's customized makefile *Makefile.hello.app*. Now you can just go to the created working directory *internal/c++/src/hello/* and build the demo application `hello.cgi` using:

```
make -f Makefile.hello_app
```

update_core.sh: Update the Portable and Core Components

Usage Summary (path note):

```
update_core.sh [--no-projects] [<dirs>]
```

Once you have obtained the core C++ Toolkit sources, with ***cvs_core.sh*** or otherwise, the local copies will become out of sync with the master CVS repository contents when other developers commit their changes. ***update_core.sh*** will update your local core source tree with any changed files without the side-effect of simultaneously checking out non-core portions of the tree. Subdirectories **/CVS* and **/internal* do not get updated by this script.

The --*no-projects* switch excludes any `Windows` or `MacOS` project files from the update. Specifically, those subdirectory names of the form **_prj* are skipped during the update when this flag is set.

The list *[<dirs>]*, when present, identifies the set of directories relative to the current directory to update. The default list of updated directories is:

- *.*
- *compilers*
- *doc*
- *include*
- *scripts*
- *src*

Note that the default list is not pushed onto a user-supplied list of directories.

update_projects.sh: Update Sources for Specific Projects

Usage Summary (path note):

```
update_projects.sh <project-list> [<directory>]
```

The script ***update_projects.sh*** facilitates work with those projects not in the core C++ tree. Because the source code and makefiles are distributed in more than one subdirectory of $CVS-ROOT */internal/c++*, this script assembles the set of required files and places them in your local C ++ source tree.

The list of projects to obtain and/or update appear in <*project-list*>, formatted according to the simple syntax used by the configure script. Project aliases and influencing how CVS recurses into subdirectories are among the options available.

This script supports two modes of operation:

1. New C++ Source Tree: If you specify a directory name, it will create the directory if necessary and check the specified portion of the C++ tree out into it, along with any additional infrastructure needed for the build system to work. (If the directory already exists, it must be empty.) It will then optionally configure and build the new tree.

2. C++ Source Tree Exists: If you run it from the top level of an existing checkout of the C++ source tree, it will update the sources and headers for the specified projects. It will then optionally reconfigure and rebuild the tree.

In either mode, if <*project-list*> contains neither a directory nor an extension, the script will add the extension *.lst* and look in the "system" directory -- which is either *internal/projects* subdir of ***update_projects.sh***'s dir or (if no such dir exists)*$NCBI/c++/scripts/internal/projects*.

New Projects and New Modules

A more complete discussion of how to begin a new project can be found on the"New Projects" page.

The following topics are discussed in this section:

- new_project.sh: Starting a New Project Outside the C++ Toolkit Tree
- Creating a New Project Inside the C++ Toolkit Tree
- Template Source Files for New Modules

new_project.sh: Starting a New Project Outside the C++ Toolkit Tree

Usage Summary (path note):

```
new_project.sh <name> <type> [builddir]
```

This script will create a startup makefile for a new, from-scratch project called *"name"* which uses the NCBI C++ Toolkit (and possibly the C Toolkit as well). For new libraries, *type=lib* while for new applications *type=app*.

Sample code will be included in the project directory for new applications. Different samples are available for *type=app[/basic]* (a command-line argument demo application based on the `corelib` library), *type=app/cgi* (for a CGI or Fast-CGI application), *type=app/objmgr* (for an application using the *Object Manager*) and *type=app/objects* (for an application using ASN.1 objects).

You will need to slightly edit the resultant makefile to:

- specify the name of your library (or application)
- specify the list of source files going to it
- modify some preprocessor, compiler, etc. flags, if needed
- modify the set of additional libraries to link to it (if it's an application), if needed

For EXAMPLE:

```
new_project.sh foo app/basic
```

creates a model makefile *Makefile.foo_app* to build an application using tools and flags hard-coded in *$NCBI/c++/Debug/build/Makefile.mk*, and headers from *$NCBI/c++/include/*. The file */tmp/foo/foo.cpp* is also created; you can either replace this with your own *foo.cpp* or modify its sample code as required.

Now, after specifying the application name, list of source files, etc., you can just go to the created working directory *foo/* and build your application using:

```
make -f Makefile.foo_app
```

You can easily change the active version of NCBI C++ Toolkit by manually setting variable `$(builddir)` in the file *Makefile.foo_app* to the desired Toolkit path, e.g., `builddir = $(NCBI)/c++/GCC-Release/build`

Creating a New Project Inside the C++ Toolkit Tree

To create your new project (e.g., "bar_proj") directories in the NCBI C++ Toolkit CVS tree (assuming that the whole NCBI C++ has been checked out to directory *foo/c++/*):

```
cd foo/c++/include && mkdir bar_proj && cvs add -m "Project Bar" bar_proj
cd foo/c++/src     && mkdir bar_proj && cvs add -m "Project Bar" bar_proj
```

From there, you can now add and edit your project C++ files.

NOTE: remember to add this new project directory to the `$(SUB_PROJ)` list of the upper level meta-makefile configurable template (e.g., for this particular case, to *foo/c++/src/Makefile.in*).

Template Source Files for New Modules

Projects in the NCBI C++ Toolkit are composed of "modules". In turn, a module is a basic logical unit of organization for the source. Usually, each C++ module consists of 2 files:

- Header file(*.hpp) (see Box 2) -- API for the external users. Ideally, this file contains only (well-commented) declarations and inline function implementations for the public interface. No less, and no more.
- Source file(*.cpp) (see Box 3) -- Definitions of non-inline functions and internally used things that should not be included to other modules.

Each and every source file **must** include NCBI disclamer and (preferably) CVS info. Then, the header file must be protected from double-includes.

The standard source templates *.hpp (see Box 2) and *.cpp (see Box 3) allow one to simply cut-and-paste to start a new C++ file (just dont forget to replace the "framewrk" stubs by your new module name). You may also use the samples checked out by ***new_project.sh*** as source file templates.

Using CVS on MS Windows in NCBI

1. You can have a pre-installed CVS executable on your PC, for example in *C:\WINNT\System32\cvs.exe*. If not, you can get it from *\\DIZZY\coremake\public\bin\cvs.exe* or *\\Basie\IEB\cvs.exe*. You also can find its latest version at http://www.cyclic.com/. Just copy it to *C:\WINNT\System32\cvs.exe*.

2. Create environment variable `CVSROOT`:

   - Click the right mouse button on the icon of your PC "*My Computer*" (it is usually situated in the upper left corner of the desktop), and then select "*Properties*" from the popup menu.
   - Form "*System Properties*" shows up. Here, choose tab "*Advanced*" and then press the button "*Environment Variables*" (users of older NT systems may instead want to choose tab "*Environment*"). Locate the part of the window titled "*User Variables for Yourname*", and then click at the end of the list the line containing variable `TEMP`.
   - Press button "*New...*".
   - Now, type `CVSROOT` in the text field "*Variable Name*", then type *:pserver:yourlogname@cvsvault.ncbi.nlm.nih.gov:/src/NCBI/vault.ncbi* in the text field "*Variable Value*". Here, the *yourlogname* stands for your NCBI account name with all letter lowercased. For example, *:pserver:vakatov@cvsvault.ncbi.nlm.nih.gov:/src/NCBI/vault.ncbi*. **NOTE:** In some cases, the *.ncbi.nlm.nih.gov* suffix needs to be dropped.
   - Press the button "*OK*" (or "*Set*"). The new variable `CVSROOT` and its value should appear in the pane "*User Variables for Yourname* ".
   - Apply the changes pressing "*OK*", "*Apply*", etc buttons until all popup windows open in the previous steps closed.
   - Logout, then login to your PC again.

3. Make sure you have your "home" directory set up -- i.e. pointed by the environment variable `HOMEPATH`. In NCBI, `HOMEPATH` is usually set to */*, which usually means *U:\* or to something like *C:\Users\YourLoginName*.

4. Create an empty file *.cvspass* in your "home" directory.

5. Execute (exactly once!) the following command (you can do so by selecting "*Run...*" from the main menu): `cvs login` You will be asked for a password -- enter the word **allowed**. This command will record your login info in the *.cvspass* file (so you would not have to login into CVS in the future, ever). If you get an authorization error, then send e-mail with the errors to cpp-core.

6. Now you are all set and can use all usual CVS commands.

Using CVS on Mac OS in NCBI

The following topics are discussed in this section:

- Installing and Configuring MacCVS Pro
- Installing and Configuring maccvs (GNU CVS port)

Installing and Configuring MacCVS Pro

1. Download MacCVS Pro 2.7d3 (the latest release version as of 2001-03-05) from http://sourceforge.net/projects/maccvspro.

2. Put the expanded folder somewhere appropriate, e.g. with other development tools. Launch the MacCVS Pro application.

3. Create a new session document by choosing [*File->New*] and saving the document.

4. Choose [*Edit->Session Settings...*]:

    - *Checkout and Update Options*
        - *Local Tree Directory*: Click *Set* and choose a folder in which to check out a CVS working directory.
        - *Merge Policy*: Select *Auto Merge Text Files and Update Binary Files*.
    - *Remote Host Information*
        - *CVS Server Settings*
            - *Server Hostname*: *cvsvault.ncbi.nlm.nih.gov*
            - *CVS Root*: */src/NCBI/vault.ncbi*

- *Authentication Method*: *Password*
    - *CVS User Name*: *username*
    - *CVS Password*: ***allowed***
- *Messages and Misc.*
    - *Message Output*
        - Checking *Automatically clear message window* is recommended, though optional.
- Click *OK*.

. Choose [*Action->Check Out Module...*]. Enter the module name and click *Check Out*.

Installing and Configuring maccvs (GNU CVS port)

FTP Retrieval

- **FTP Download Now** [ftp://ftp.ncbi.nih.gov/toolbox/ncbi_tools++/CURRENT/]
- **Available FTP Archives**: Select the archive for your system. When the dialog box appears, choose the destination in your filesystem for the downloaded archive. **Note:** With some browsers, you may need to right-click-and-hold with your mouse and use the *'Save Link As...', 'Copy to Folder...'*, or similar options from the drop-down menu to properly save the archive. For a current list of the source code archives for different operating system/ compiler combinations consult the current Release Notes available at *ftp://ftp.ncbi.nih.gov/ toolbox/ncbi_tools++/CURRENT/RELEASE_NOTES.html* [ftp://ftp.ncbi.nih.gov/toolbox/ ncbi_tools++/CURRENT/RELEASE_NOTES.html]
- **Unpack the Source Archive**
    - Unix SystemsThe Unix distributions have been archived using the standard *tar* command and compressed using *gzip*. When unpacked, all files will be under the directory *ncbi_cxx*, which will be created in the current directory.(**Caution:** If *ncbi_cxx* already exists, *tar* extraction will overwrite existing files.)To unpack the the archive:

```
gunzip -c ncbi_cxx_*.tar.gz | tar xvf -
```

- Windows Systems The Microsoft Windows versions of the source distribution have been prepared as self-extracting executables. By default a sub-folder *ncbi_cxx* will be created in the current folder to contain the extracted source. If *ncbi_cxx* already exists in the folder where the executable is launched, user confirmation is required before files are overwritten.To actually perform the extraction, do one of the following:

    i. Double-click on the archive's icon to create *ncbi_cxx* in the current folder.

    ii. Right-click on the archive's icon, and select *'Extract to...'* to unpack the archive to a user-specified location in the filesystem.

- Macintosh Systems The Macintosh version of the source distribution has been prepared as a 'Stuff-It' archive, and is usable from within Metrowerks CodeWarrior. The sources will be deployed to a directory *ncbi_cxx* created in the current folder.Further Mac-specific details are forthcoming.

Source Tree Structure Summary

To summarize the Getting Started page, the *source tree* is organized as follows:

- The top-level has configuration files and the directories *include/, src/, scripts/, compilers/* and *doc/*
- The *src* and *include* directories contain "projects" as subdirectories. Projects may contain sub-projects in a hierarchical fashion.
- *src/* additionally contains *makefile* and *meta-makefile* templates.
- Projects contain "modules" and various customized makefiles and meta-makefiles to control their compilation.

Box 1: Usage

cvs_core.sh Usage

```
Usage:  cvs_core.sh <dir> [--without-cvs] [--with-objects] .... [--<platform>]
Synopsis:
   Checkout a portable part of the NCBI C++ tree.
Arguments:
   <dir>              -- path to checkout the tree to
   --without-cvs      -- do not put CVS-related stuff to the resultant tree
   --with-objects     -- generate ASN.1 serialization code in the "objects/" dir
   --without-objects -- do not even checkout "objects/" dirs
   --without-ctools  -- do not checkout projects that depend on the C Toolkit
   --without-gui      -- do not checkout projects that depend on the wxWindows
   --with-vc-cfgs[={all|inhouse|none}] -- multi-config MSVC project files
   --without-dizzy    -- remove paths to \\DIZZY\ from MSVC project files
   --<platform>       -- "--unix", "--msvc", "--cygwin", "--mac", "--all"
Default:
   cvs_core.sh <dir> --with-cvs --with-ctools --with-gui --all
Note:  Must specify target directory (name cannot start with '-')
```

import_project.sh Usage

```
USAGE:  import_project.sh <cvs_tree_path> [builddir]
SYNOPSIS:
   Retrieve project (and all sub-projects) located in the NCBI C++ Toolkit
   CVS tree at:
      internal/c++/src/<cvs_tree_path>
      internal/c++/include/<cvs_tree_path>
   Create makefiles "Makefile.*_{lib,app}" (based on the original project
   makefiles "Makefile.*.{lib,app}") to build libs/apps using pre-built NCBI
   C++ Toolkit from [builddir]  (default: /netopt/ncbi_tools/c++/Debug/build);
also create
   top-level makefiles "Makefile" from "Makefile.in".
LIMITATIONS:
   - Supports only very basic substitutions in Makefile.in.
   - May produce bogus library search paths.
```

new_project.sh Usage

```
USAGE:  new_project.sh <name> <type> [builddir]
SYNOPSIS:
   Create a model makefile "Makefile.<name>_<type>" to build
   a library or an application that uses pre-built NCBI C++ toolkit.
   Also include sample code when creating applications.
ARGUMENTS:
   <name>      -- name of the project (will be subst. to the makefile name)
   <type>      -- one of the following:
                  lib         to build a library
```

```
                   app[/basic] to build a simple application
                   app/cgi     to build a CGI or FastCGI application
                   app/objects to build an application using ASN.1 objects
                   app/objmgr  to build an application using the object manager
    [builddir]  -- path to the pre-built NCBI C++ toolkit
                   (default = /netopt/ncbi_tools/c++/Debug/build)
```

update_core.sh **Usage**

```
Usage: update_core.sh [--no-projects] [--help] [<dirs>]
  --no-projects: Do not update Windows/MacOS project files.
  --help:        Print this message.
  <dirs>: The list of subdirectories to update; "." updates the files in the
  top-level directory.  If no directories are specified, the default list is
    . compilers doc include scripts src
```

update_projects.sh **Usage**

```
Usage: update_projects.sh <project-list> [<directory>]
This script supports two modes of operation:
  * If you specify a directory name, it will create the directory if
    necessary and check the specified portion of the C++ tree out into
    it, along with any additional infrastructure needed for the build
    system to work.  (If the directory already exists, it must be
    empty.)  It will then optionally configure and build the new tree.
  * If you run it from the top level of an existing checkout of the
    C++ source tree, it will update the sources and headers for the
    specified projects.  It will then optionally reconfigure and
    rebuild the tree.
The syntax for project lists is documented at:
http://www.ncbi.nlm.nih.gov/IEB/ToolBox/CPP_DOC/config.html#ref_ProjectListIn
either mode, if  contains neither a directory nor an
extension, the script will add the extension .lst and look in the
system directory ".../c++/scripts/internal/projects".
```

Box 2: Source File Template (.hpp)

```
#ifndef FRAMEWRK__HPP
#define FRAMEWRK__HPP

/*  $Id: ch_getcode.xml,v 1.45 2004/03/23 16:57:09 siyan Exp $
* ===========================================================================
*
*                            PUBLIC DOMAIN NOTICE
*               National Center for Biotechnology Information
*
*  This software/database is a "United States Government Work" under the
*  terms of the United States Copyright Act.  It was written as part of
*  the author's official duties as a United States Government employee and
*  thus cannot be copyrighted.  This software/database is freely available
*  to the public for use. The National Library of Medicine and the U.S.
*  Government have not placed any restriction on its use or reproduction.
*
*  Although all reasonable efforts have been taken to ensure the accuracy
*  and reliability of the software and data, the NLM and the U.S.
*  Government do not and cannot warrant the performance or results that
*  may be obtained by using this software or data. The NLM and the U.S.
*  Government disclaim all warranties, express or implied, including
*  warranties of performance, merchantability or fitness for any particular
*  purpose.
*
*  Please cite the author in any work or product based on this material.
*
* ===========================================================================
*
* Author:  !!! PUT YOUR NAME(s) HERE !!!
*
* File Description:
*   !!! PUT YOUR DESCRIPTION HERE !!!
*
*/

#include <corelib/ncbistd.hpp>
//  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
//  !!! PUT YOUR OTHER #include's HERE !!!
//  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!


// (BEGIN_NCBI_SCOPE must be followed by END_NCBI_SCOPE later in this file)
BEGIN_NCBI_SCOPE


//  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
//  !!! PUT YOUR CODE(PROTOTYPES, TYPEDEFS, ETC.) HERE !!!
//  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
//  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
//  !!! FINALLY, IMPLEMENT YOUR INLINE FUNCTIONS HERE !!!
//  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!


// (END_NCBI_SCOPE must be preceded by BEGIN_NCBI_SCOPE)
END_NCBI_SCOPE

#endif  /* FRAMEWRK__HPP */
```

Box 3: Source File Template (.cpp)

```
/*  $Id: ch_getcode.xml,v 1.45 2004/03/23 16:57:09 siyan Exp $
* ===========================================================================
*
*                            PUBLIC DOMAIN NOTICE
*               National Center for Biotechnology Information
*
*  This software/database is a "United States Government Work" under the
*  terms of the United States Copyright Act.  It was written as part of
*  the author's official duties as a United States Government employee and
*  thus cannot be copyrighted.  This software/database is freely available
*  to the public for use. The National Library of Medicine and the U.S.
*  Government have not placed any restriction on its use or reproduction.
*
*  Although all reasonable efforts have been taken to ensure the accuracy
*  and reliability of the software and data, the NLM and the U.S.
*  Government do not and cannot warrant the performance or results that
*  may be obtained by using this software or data. The NLM and the U.S.
*  Government disclaim all warranties, express or implied, including
*  warranties of performance, merchantability or fitness for any particular
*  purpose.
*
*  Please cite the author in any work or product based on this material.
*
* ===========================================================================
*
* Author:  !!! PUT YOUR NAME(s) HERE !!!
*
* File Description:
*   !!! PUT YOUR DESCRIPTION HERE !!!
*
* ===========================================================================
*/

#include <corelib/ncbistd.hpp>
#include <foo_proj_dir/framewrk.hpp>
//  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
//  !!! PUT YOUR OTHER #include's HERE !!!
//  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!


// This is to use the ANSI C++ standard templates without the "std::" prefix
// NCBI_USING_NAMESPACE_STD;

// This is to use the ANSI C++ standard templates without the "std::" prefix
// and to use NCBI C++ entities without the "ncbi::" prefix
// USING_NCBI_SCOPE;

// (BEGIN_NCBI_SCOPE must be followed by END_NCBI_SCOPE later in this file)
BEGIN_NCBI_SCOPE
```

```
//  !!!!!!!!!!!!!!!!!!!!!!!!!!
//  !!! PUT YOUR CODE HERE !!!
//  !!!!!!!!!!!!!!!!!!!!!!!!!!


// (END_NCBI_SCOPE must be preceded by BEGIN_NCBI_SCOPE)
END_NCBI_SCOPE
```